



UNIVERSITÀ
DEGLI STUDI DI TRIESTE



Sequential statements

A.Carini – Progettazione di sistemi elettronici

IF statement

```
if_statement ←  
  [ if_label : ]  
  if condition then  
    { sequential_statement }  
  { elsif condition then  
    { sequential_statement } }  
  [ else  
    { sequential_statement } ]  
  end if [ if_label ] ;
```

IF statement examples

```
if sel = 0 then  
    result <= input_0;    -- executed if sel = 0  
else  
    result <= input_1;    -- executed if sel /= 0  
end if;
```

```
if mode = immediate then  
    operand := immed_operand;  
elsif opcode = load or opcode = add or opcode = subtract then  
    operand := memory_operand;  
else  
    operand := address_operand;  
end if;
```

IF statement examples

```
if opcode = halt_opcode then
    PC := effective_address;
    executing := false;
    halt_indicator <= true;
end if;
```

```
if phase = wash then
    if cycle_select = delicate_cycle then
        agitator_speed <= slow;
    else
        agitator_speed <= fast;
    end if;
    agitator_on <= true;
end if;
```

A thermostat control

```
entity thermostat is
    port ( desired_temp, actual_temp : in integer;
          heater_on : out boolean );
end entity thermostat;

-----

architecture example of thermostat is
begin
    controller : process (desired_temp, actual_temp) is
    begin
        if actual_temp < desired_temp - 2 then
            heater_on <= true;
        elsif actual_temp > desired_temp + 2 then
            heater_on <= false;
        end if;
    end process controller;
end architecture example;
```

Conditional variable assignment (VHDL 2008)

```
conditional_variable_assignment ←  
  [ label : ]  
  name := expression when condition  
    { else expression when condition }  
    [ else expression ] ;
```

- Example:

```
result := a - b when mode = subtract else a + b;
```

- Which is equivalent to

```
if mode = subtract then  
  result := a - b;  
else  
  result := a + b;  
end if;
```

Case statement

```
case_statement ←  
  [ case_label : ]  
  case expression is  
    ( when choices => { sequential_statement } )  
    { ... }  
  end case [ case_label ] ;  
choices ← ( simple_expression | discrete_range | others ) { | ... }
```

Case statement example

```
type alu_func is (pass1, pass2, add, subtract);
```

```
case func is
```

```
  when pass1 =>
```

```
    result := operand1;
```

```
  when pass2 =>
```

```
    result := operand2;
```

```
  when add =>
```

```
    result := operand1 + operand2;
```

```
  when subtract =>
```

```
    result := operand1 - operand2;
```

```
end case;
```


Case statement example

```
subtype index_mode is integer range 0 to 3;  
variable instruction_register : integer range 0 to 2**16 - 1;
```

```
case index_mode'((instruction_register / 2**12) rem 2**2) is  
  when 0 =>  
    index_value := 0;  
  when 1 =>  
    index_value := accumulator_A;  
  when 2 =>  
    index_value := accumulator_B;  
  when 3 =>  
    index_value := index_register;  
end case;
```

Multiple choices, others

```
type opcodes is  
    (nop, add, subtract, load, store, jump, jumpsub, branch, halt);
```

```
case opcode is  
    when load | add | subtract =>  
        operand := memory_operand;  
    when store | jump | jumpsub | branch =>  
        operand := address_operand;  
    when others =>  
        operand := 0;  
end case;
```

Discrete range in choices

```
discrete_range ←  
  discrete_subtype_indication  
  [ simple_expression ( to [ downto ] simple_expression  
subtype_indication ←  
  type_mark  
  [ range simple_expression ( to [ downto ] simple_expression ]
```

Example:

```
case opcode is  
  when add to load =>  
    operand := memory_operand;  
  when branch downto store =>  
    operand := address_operand;  
  when others =>  
    operand := 0;  
end case;
```

Discrete range in choices

- With a subtype:

```
subtype control_transfer_opcodes is opcodes range jump to branch;
```

```
when control_transfer_opcodes | store =>  
    operand := address_operand;
```

Choices: must be locally static

```
variable N : integer := 1;  
  
case expression is      -- example of an illegal case statement  
  when N | N+1 => ...  
  when N+2 to N+5 => ...  
  when others => ...  
end case;
```

LEGAL USE:

```
constant C : integer := 1;
```

```
case expression is  
  when C | C+1 => ...  
  when C+2 to C+5 => ...  
  when others => ...  
end case;
```

Summarizing

- All possible values of the selector must be covered by a single choice.
- The values of choices must be locally static.
- The choice **others**, if present, must be the last and there can be only one choice **others**.

Example: a multiplexer

```
type sel_range is range 0 to 3;
```

```
library ieee; use ieee.std_logic_1164.all;  
entity mux4 is  
    port ( sel : in sel_range;  
          d0, d1, d2, d3 : in std_ulogic;  
          z : out std_ulogic );  
end entity mux4;
```

Example: a multiplexer

```
architecture demo of mux4 is
begin
  out_select : process (sel, d0, d1, d2, d3) is
  begin
    case sel is
      when 0 =>
        z <= d0;

      when 1 =>
        z <= d1;
      when 2 =>
        z <= d2;
      when 3 =>
        z <= d3;
    end case;
  end process out_select;
end architecture demo;
```


Selected variable assignment (VHDL 2008)

```
selected_variable_assignment <=
  [ label : ]
  with expression select
    name := { expression when choices , }
           expression when choices ;
```

Example:

```
with func select
  result := operand1           when pass1,
           operand2           when pass2,
           operand1 + operand2 when add,
           operand1 - operand2 when subtract;
```

NULL statement

```
null_statement  $\Leftarrow$  [ label : ] null ;
```

Example:

```
case opcode is  
  when add =>  
    Acc := Acc + operand;  
  when subtract =>  
    Acc := Acc - operand;  
  when nop =>  
    null;  
end case;
```

NULL statement

Another example:

```
control_section : process ( sensitivity-list ) is  
begin  
    null;  
end process control_section;
```

Infinite loop

```
loop_statement ←  
  [ loop_label : ]  
  loop  
    { sequential_statement }  
  end loop [ loop_label ] ;
```

A counter

```
entity counter is  
    port ( clk : in bit; count : out natural );  
end entity counter;  
  
-----  
architecture behavior of counter is  
begin  
    incrementer : process is  
        variable count_value : natural := 0;  
    begin  
        count <= count_value;  
        loop  
            wait until clk = '1';  
            count_value := (count_value + 1) mod 16;  
            count <= count_value;  
        end loop;  
    end process incrementer;  
end architecture behavior;
```

Exit statement

```
exit_statement ←  
  [ label : ] exit [ loop_label ] [ when boolean_expression ] ;
```

Esempi:

```
exit;
```

```
if condition then  
  exit;  
end if;
```



```
exit when condition;
```

A reset counter

```
entity counter is
    port ( clk, reset : in bit; count : out natural );
end entity counter;

-----

architecture behavior of counter is
begin
    incrementer : process is
        variable count_value : natural := 0;
    begin
        count <= count_value;
        loop
            loop
                wait until clk = '1' or reset = '1';
                exit when reset = '1';
                count_value := (count_value + 1) mod 16;
                count <= count_value;
            end loop;
            -- at this point, reset = '1'
            count_value := 0;
            count <= count_value;
            wait until reset = '0';
        end loop;
    end process incrementer;
end architecture behavior;
```

Label and exit

```
outer : loop
  ...
  inner : loop
    ...
    exit outer when condition-1; -- exit 1
    ...
    exit when condition-2;      -- exit 2
    ...
  end loop inner;
  ...
  exit outer when condition-3;  -- exit 3
  ...
end loop outer;
...
-- target B
```


Next statement

```
next_statement ←  
  [ label : ] next [ loop_label ] [ when boolean_expression ] ;
```

Examples:

```
next;
```

```
next when condition;
```

```
next loop-label;
```

```
next loop-label when condition;
```

While loop

```
loop_statement ←  
  [ loop_label : ]  
  while boolean_expression loop  
    { sequential_statement }  
  end loop [ loop_label ] ;
```

- Everything we said about loops hold also for while loops (also about the used of **next** and **exit**).

Example

$$\cos \theta = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \dots$$

```
entity cos is
  port ( theta : in real; result : out real );
end entity cos;
```

Example

```
architecture series of cos is
begin
    summation : process (theta) is
        variable sum, term : real;
        variable n : natural;
    begin
        sum := 1.0;
        term := 1.0;
        n := 0;
        while abs term > abs (sum / 1.0E6) loop
            n := n + 2;
            term := (-term) * theta**2 / real(((n-1) * n));
            sum := sum + term;
        end loop;
        result <= sum;
    end process summation;
end architecture series;
```

For loop

```
loop_statement ←  
  [ loop_label : ]  
  for identifier in discrete_range loop  
    { sequential_statement }  
  end loop [ loop_label ] ;
```

Discrete range is:

```
simple_expression ( to ↓ downto ) simple_expression
```

or a subtype.

Examples

```
for count_value in 0 to 127 loop
    count_out <= count_value;
    wait for 5 ns;
end loop;
```

```
type controller_state is (initial, idle, active, error);
```

```
for state in controller_state loop
    ...
end loop;
```

Loop parameter

- It is a constant inside the loop: you cannot assign values to the loop parameter!
- You do not need to declare it, it is declared by the loop itself.
- It does not exist outside the loop.
- It hides any other object with the same name.

```
erroneous : process is
  variable i, j : integer;
begin
  i := loop_param;           -- error!
  for loop_param in 1 to 10 loop
    loop_param := 5;        -- error!
  end loop;
  j := loop_param;         -- error!
end process erroneous;
```

Hiding example

```
hiding_example : process is
  variable a, b : integer;
begin
  a := 10;
  for a in 0 to 7 loop
    b := a;
  end loop;
  -- a = 10, and b = 7
  ...
end process hiding_example;
```


Example

```
architecture fixed_length_series of cos is
begin
    summation : process (theta) is
        variable sum, term : real;
    begin
        sum := 1.0;
        term := 1.0;
        for n in 1 to 9 loop
            term := (-term) * theta**2 / real(((2*n-1) * 2*n));
            sum := sum + term;
        end loop;
        result <= sum;
    end process summation;
end architecture fixed_length_series;
```

Syntax of loop

```
loop_statement ←  
  [ loop_label : ]  
  [ while boolean_expression | for identifier in discrete_range ] loop  
    { sequential_statement }  
  end loop [ loop_label ] ;
```

Assert and report

```
assertion_statement ←  
    [ label : ] assert boolean_expression  
        [ report expression ] [ severity expression ] ;
```

Examples:

```
assert initial_value <= max_value;
```

```
assert initial_value <= max_value  
    report "initial value too large";
```

```
assert current_character >= '0' and current_character <= '9'  
    report "Input number " & input_string & " contains a non-digit";
```

Severity level

```
type severity_level is (note, warning, error, failure);
```

Examples:

```
assert free_memory >= low_water_limit  
  report "low on memory, about to start garbage collect"  
  severity note;
```

```
assert packet_length /= 0  
  report "empty network packet received"  
  severity warning;
```

```
assert clock_pulse_width >= min_clock_width  
  severity error;
```

```
assert (last_position - first_position + 1) = number_of_entries  
  report "inconsistency in buffer model"  
  severity failure;
```

Example latch SR

```
entity SR_flipflop is
    port ( S, R : in bit; Q : out bit );
end entity SR_flipflop;

-----

architecture checking of SR_flipflop is
begin
    set_reset : process (S, R) is
    begin
        assert S = '1' nand R = '1';
        if S = '1' then
            Q <= '1';
        end if;
        if R = '1' then
            Q <= '0';
        end if;
    end process set_reset;
end architecture checking;
```

Example edge triggered register

```
entity edge_triggered_register is
  port ( clock : in bit;
        d_in : in real; d_out : out real );
end entity edge_triggered_register;

-----

architecture check_timing of edge_triggered_register is
begin
  store_and_check : process (clock) is
    variable stored_value : real;
    variable pulse_start : time;
  begin
    case clock is
      when '1' =>
        pulse_start := now;
        stored_value := d_in;
        d_out <= stored_value;
      when '0' =>
        assert now = 0 ns or (now - pulse_start) >= 5 ns
          report "clock pulse too short";
    end case;
  end process store_and_check;
end architecture check_timing;
```

Report statement

```
report_statement ←  
  [ label : ] report expression [ severity expression ] ;
```

Example:

```
transmit_element : process (transmit_data) is  
  ...      -- variable declarations  
begin  
  report "transmit_element: data = "  
    & data_type'image(transmit_data);  
  ...  
end process transmit_element;
```

See:

- Peter Ashenden, «The designers' guide to VHDL» Morgan Kaufmann,
 - Chapter 3